Algorytm genetyczny w szeregowaniu zadań.

Kamil Piotrowski 122491, Michał Lewiński 122505

14.01.2016r.

kontakt: kamil.piotrowski@student.cs.put.poznan.pl

michal.lewinski@student.cs.put.poznan.pl

1 Wstęp

Niniejsze sprawozdanie dotyczy problemu nr 3 związanego z szeregowaniem zadań (job shop, liczba maszyn $m = 2$, operacje typu non-preemptive, dla pierwszej maszyny od $k = 2$ do $n/2$ okresów przestoju o losowym czasie rozpoczęcia i trwania, różne i losowe czasy gotowości dla wszystkich zadań, minimalizacja sumy czasów zakończenia wszystkich operacji, liczba zadań n). Do wykonania tego problemu wykorzystany został algorytm genetyczny, który stosuje mutacje i krzyżowania przy rozszerzaniu populacji. Wartość funkcji celu określana jest jako suma czasów zakończenia operacji. Sprawozdanie składa się z czterech rozdziałów: pierwszy rozdział zawiera teoretyczny opis działania i implementacji naszego algorytmu. W drugim rozdziale znajdują się testy dotyczące strojenia heurystyki, na podstawie, których zostaną wybrane najoptymalniejsze ustawienia. W trzecim rozdziale znajdują się testy instancji. W ostatnim czwartym rozdziale zamieszczone zostały wnioski i podsumowanie przeprowadzonych badań.

1.1 Schemat algorytmu

Algorytm genetyczny, który został wykorzystany w badaniach opiera się na mutacji i krzyżowaniu. W początkowej fazie działania, program generuje instancję problemu na podstawie wartości poszczególnych parametrów, które mogą zostać także zdefiniowane przez użytkownika. Tworzona jest początkowa populacja, której rozmiar jest zdefiniowany na początku algorytmu (domyślnie 50). Następnie populacja zostaje rozszerzona za pomocą mutacji, krzyżowania oraz łączonej mutacji z krzyżowaniem. Liczba poszczególnych szeregowań tworzona za pomocą wymienionych przekształceń także jest określana przez odpowiednie parametry procentowe, które w sumie muszą utworzyć populację końcową $liczba\,mutacji + liczba\,krzyżowań + liczba\,łączonych\,(mutacja + krzyżowanie) = 1$ (domyślnie liczba poszczególnych operacji jest równa, a rozmiar końcowej populacji domyślnie wynosi 300). Uzyskana w ten sposób populacja poddawana jest selekcji za pomocą turnieju, w którym eliminowane są gorsze uszeregowania, tak aby rozmiar zredukowanej populacji wyniósł tyle samo co początkowej. Powyższe przekształcenia są wykonywane tak długo aż nie zostaną spełnione kryteria zakończenia algorytmu, w tym przypadku jest to czas działania całego algorytmu. Cały proces algorytmu został przedstawiony na schemacie przedstawionym poniżej.

Rysunek 1: Uproszczony schemat algorytmu

1.2 Generator instancji

Generator instancji tworzy losowe rozwiązania tak aby uzyskać początkową populacje o określonym rozmiarze. Poszczególne rozwiązania powstają na podstawie ilości zadań n oraz przerw p. Liczba przerw wyznaczana jest na podstawie wzoru (1), gdzie n to liczba zadań, natomiast $procentP$ oznacza stosunek ilości przerw do zadań.

Zadanie składa się z dwóch podzadań. Każde podzadanie zawiera: nr maszyny na której ma się wykonać; informacje czy zostało już dodane do uszeregowania; czas, od którego podzadanie może zacząć się wykonywać; które podzadanie jest z nim w parzę (referencja do drugiego podzadania) oraz czy musi zostać wykonane jako pierwsze czy drugie w kolejności (numer operacji). Przerwy znajdują się wyłącznie na pierwszej maszynie i ich czas rozpoczęcia nie może ulec zmianie w trakcie działania algorytmu.

Po utworzeniu listy zadań i przerw dochodzi do ich posortowania. Lista zadań sortowana jest na podstawie czasu gotowości zadania, natomiast lista przerw na podstawie czasu rozpoczęcia.

$$p = procentP * n + 2 \tag{1}$$

1.3 Generator losowego rozwiązania

Generator losowego rozwiązania tworzy rozwiązanie na podstawie listy zadań i przerw utworzonych wcześniej przez generator instancji. Rozwiązanie oparte jest na uszeregowaniu dwóch maszyn. Uszeregowanie pojedyńczej maszyny to lista składająca się z bloków, które posiadają czas startu Ts, czas końca Tk i czas trwania Tt. Każdy poszczególny blok może być zadaniem lub przerwą.Każde uszeregowanie rozpoczyna się od bloku pomocniczego, którego czasy Ts, Tk i Tt wynosi 0. Ten początkowy blok ma pomóc przy dodawaniu kolejnych przerw i zadań, gdyż nie zależnie od miejsca, mechanizm dodawania bloku będzie taki sam. Na początku tworzenia rozwiązania dodawane są wszystkie przerwy do pierwszej maszyny, gdyż ich czas początkowy nie może ulec zmianie. Następnie dla maszyny pierwszej i drugiej wylosowane są podzadania ze zbioru operacji, które muszą się wykonać jako pierwsze oraz operacji nr 2, dla których wykonała się operacja nr 1. . W momencie dodawania do uszeregowania sprawdzone zostaje, czy w lukach między blokami (zaczynając od początku uszeregowania) może zmieścić się podzadanie $Ts_{n+1} - Tk_n \geq Tt_k$, gdzie k to dodawane zadanie. Jeżeli podzadanie było pierwsze to sprawdzone zostaje ponadto, czy czas gotowości będzie równy lub mniejszy niż Tk_n. Jeżeli nie, to sprawdzone zostaje czy podzadanie będzie gotowe w powstałej luce i czy wciąż będzie się mieścić. Natomiast jeżeli podzadanie ma zostać wykonane jako drugie to musi zostać sprawdzone, czy Tk pierwszego podzadania jest mniejsze bądź równe Ts dodawanego podzadania. Jeżeli podzadanie nie mieści się w znalezionej luce między przerwami, spraw-

dzana zostaje następna luka, a w najgorszym wypadku operacja ta ląduje na końcu obecnego uszeregowania.

1.4 Mutacja

Mutacja jest metodą, której celem jest dokonanie pewnej losowej zmiany w uszeregowaniu. Kluczową cechą mutacji jest jej siła SM, która determinuje jak bardzo rozwiązanie jest modyfikowane. Pełna mutacja, w której z rozwiązania macierzystego otrzymujemy potomka opiera się na kilku pojedynczych mutacjach wykonywanych na każdej maszynie w ilości K, opisanej wzorem (2). Siła mutacji ulega redukcji co 1 sekundę działania algorytmu $SM = SM/1,5$.

$$K = \lceil N * SM \rceil \tag{2}$$

$$L = (-1) * \left[\left\lceil \frac{N * SM}{2} \right\rceil + 1 \right] \tag{3}$$

Zmiana uszeregowaniu, której dokonuje pojedyncza mutacja opera się na przesunięciu losowego elementu o L (3) pozycji w lewo. Gdy indeks elementu jest mniejszy od wartości przesunięcia element, jest on umieszczany na pierwszym miejscu w uszeregowaniu. Po wykonaniu przesunięcia następuje naprawa uszeregowań na każdej z maszyn, która opiera się na sprawdzeniu, czy zadania nie nachodzą na siebie oraz są wykonywane w odpowiednim czasie (dla pierwszej operacji jest to czas gotowości natomiast dla operacji drugiej czas Tk operacji pierwszej). Naprawa opiera się głównie na przesunięciu problematycznych operacji w prawo. Gdy przesuwana operacja nachodzi na przerwę konserwującą to miejsce operacji zostaje zamienione z przerwą i algorytm naprawy jest kontynuowany. Algorytm trwa tak długo aż wszystkie operacje znajdą się w odpowiednich miejscach.

1.5 Krzyżowanie

Krzyżowanie jest to metoda, której celem jest modyfikacja istniejących rozwiązań na podstawie innych uszeregowań. W pierwszej fazie, maszyny są dzielone na dwie równe połowy na podstawie liczby podzadań. Jedna połowa uszeregowania pozostaje niezmieniona, natomiast druga trafia do listy tymczasowej. Na podstawie ułożenia podzadań w innym rozwiązaniu, operacje w liście tymczasowej dodawane są po kolei, aż uszeregowanie będzie całkowicie uzupełnione. Podczas wykonywania krzyżowania, połowa otrzymywanych rozwiązań pozostawia pierwszą połowę nie zmienioną, natomiast reszta drugą połowę. Podobnie jak w mutacji, po wykonaniu krzyżowania następuje naprawienie uszeregowania. Sprawdzone zostaje, czy operacje nie nachodzą na siebie i czy podzadanie pierwsze wykonało się przed rozpoczęciem podzadania drugiego. Krzyżowanie nie korzysta z żadnych dodatkowych parametrów ze względu na to, że punkt dzielenia krzyżowania jest tylko po środku całego uszeregowania.

1.6 Selekcja

Selekcja końcowej populacji opiera się na turnieju, w którym liczba porównywanych rozwiązań wynosi $\frac{populacja\ końcowa}{populacja\ początkowa}$. Selekcja wykonywana jest tak długo aż rozmiar zredukowanej populacji wyniesie tyle samo co rozmiar populacji początkowej. Taki sposób selekcji ma wadę, gdyż mogą zostać utracone najgorsze rozwiązania w momencie, gdy rozwiązania ułożone są rosnąco. Zawsze po wykonaniu selekcji, porzucane jest rozwiązanie „najgorsze". Zdecydowaliśmy się na algorytm turnieju ze względu na małą szansę wpadnięcia heurystyki w optimum lokalne niż w przypadku wzięcia wszystkich najlepszych rozwiązań.

2 Testy algorytmu

W celu poprawy działania algorytmu, przeprowadziliśmy testy, które miały umożliwić nam znalezienie odpowiednich wartości parametrów. Aby zwiększyć prawidłowość otrzymanych wyników, wartości zostały obliczone za pomocą średniej z 200 instancji (jeden punkt pomiarowy to średnia z 200 rozwiązań). Testy przeprowadzaliśmy na tych samych ustawieniach generatora instancji, które zostały przedstawione w tabeli instancji 1. Parametry, których wielkości były optymalizowane, zostały przedstawione w tabeli 2.

minZ	minimalny czas trwania pojedynczej operacji
maxZ	maksymalny czas trwania pojenyńczej operacji
N	liczba zadań
maxG	maksymalny czas gotowości dla zadania
PP	liczba przerw konserwujących [(% z N)+2]
minP	minimalny czas trwania przerwy konserwującej
maxP	maksymalny czas trwania przerwy konserwującej

Tablica 1: opis parametrów instancji

T	czas działania algorytmu
SM	siła mutacji
PS	liczba populacji początkowej
PE	liczba populacji końcowej
IM	liczba mutacji
IK	liczba krzyżowania

Tablica 2: opis parametrów wykorzystanych w testach

parametr	minZ	maxZ	N	maxG	PP	minP	maxP
wartość	1	50	50	100	0,2	1	50

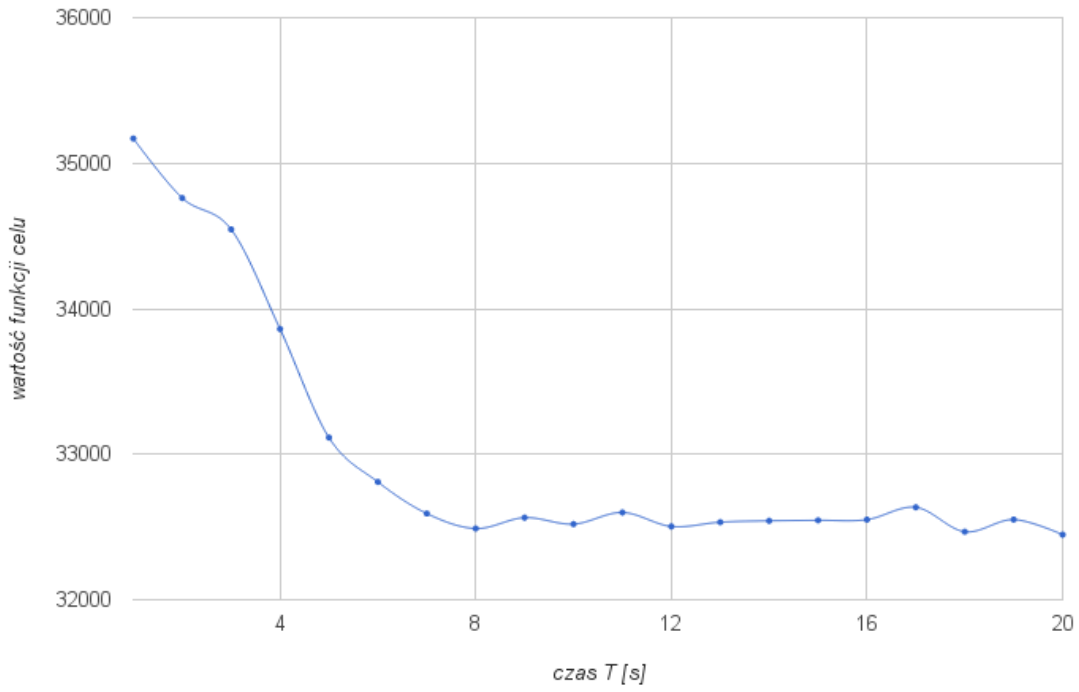
Tablica 3: ustawienia generatora instancji dla strojenia

2.1 Test wartości funkcji celu od czasu przed strojeniem

Test miał za zadanie sprawdzić, jak zmienia się optymalizowana wartość w funkcji czasu. Z otrzymanych wyników można wywnioskować, że czas optymalizacji uszeregowań stabilizuje się po $8s$ działania algorytmu. Test ten pozwolił wyznaczyć wartość T, która będzie wykorzystywana w następnych testach (jej zwiększenie nie powoduje poprawy rozwiązania).

parametr	SM	PS	PE	IM	IK	T
wartość	0,5	50	300	0,33	0,33	$0 - 20s$

Tablica 4: tabela użytych parametrów dla testu 2.1



Rysunek 2: wykres dla testu wartości funkcji celu od czasu przed strojeniem

2.2 Test siły mutacji

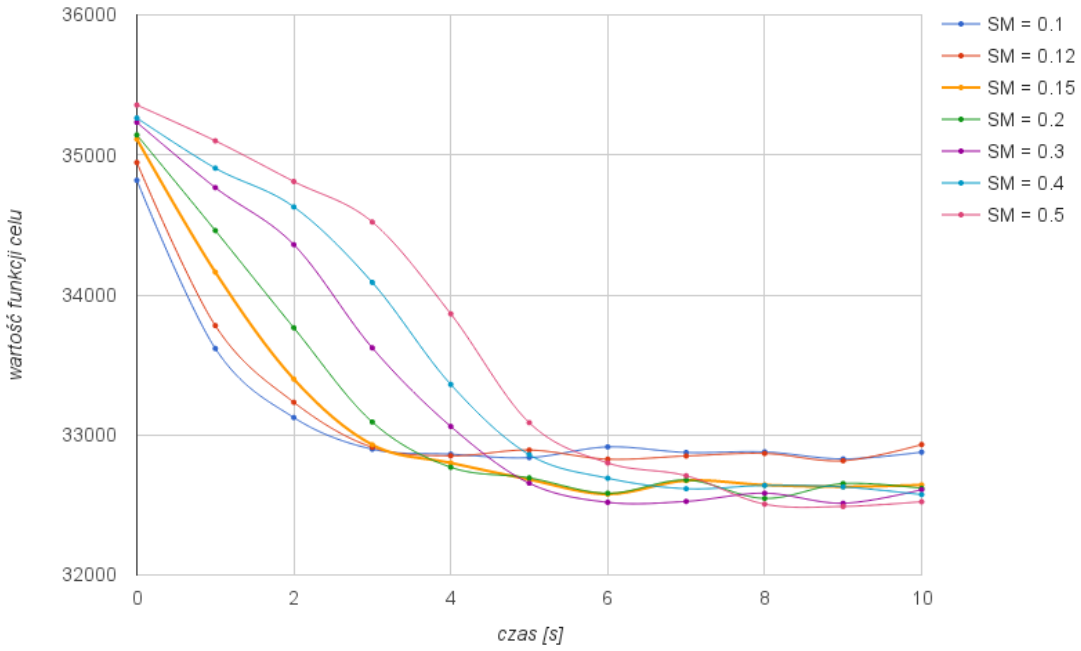
Celem tego testu było sprawdzenie jak siła mutacji wpływa na szybkość optymalizowania rozwiązania przez algorytm. Występuje tutaj bardzo ciekawa zależność. Mutacja o dużej sile początkowo optymalizuje rozwiązanie wolniej ze względu na dużą liczbę zmian w nim dokonywanych. Po pewnym czasie gdy siła mutacji jest zmniejszana na wykresie widzimy większy spadek krzywej. Bardzo słaba mutacja ma natomiast tendencję do wpadanie w optimum lokalne ze względu na małą liczbę zmian dokonywanych w uszeregowaniu. Na podstawie tego testu wyznaczyliśmy optymalną siłę mutacji na $0,15$, która będzie wykorzystywana w następnych testach.

parametr	SM	PS	PE	IM	IK	T
wartość	$0,1 - 0,5$	50	300	$0,33$	$0,33$	$0 - 10s$

Tablica 5: tabela użytych parametrów dla testu 2.2

2.3 Test wielkości populacji początkowej i końcowej

Test ten miał za zadanie oszacować jak wartość funkcji celu zależy od wielkości populacji początkowej i końcowej. Należy tutaj dodać, że jeżeli kończymy program na podstawie czasu, zwiększenie wielkości populacji będzie skutkować zmniejszeniem iteracji algorytmu. Optymalnym rozwiązaniem okazało się wybranie populacji początkowej równej 250 oraz końcowej równej 750. Z wykresu możemy także wywnioskować, że najgorsze wyniki otrzymujemy zaczynając od małej populacji początkowej ($PS = 50$) lub bardzo dużej populacji końcowej



Rysunek 3: wykres dla testu siły mutacji

$(PE = 8PS)$.

parametr	SM	PS	PE	IM	IK	T
wartość	$0,15$	$50-300$	$[3-8]*PS$	$0,33$	$0,33$	$8s$

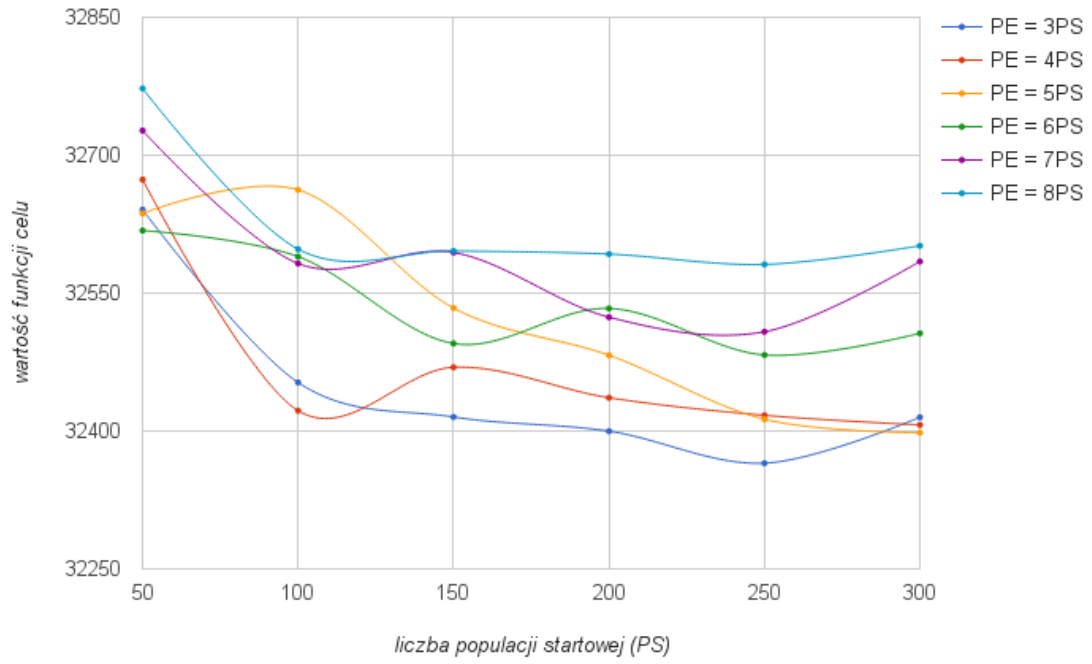
Tablica 6: tabela użytych parametrów dla testu 2.3

2.4 Test liczby mutacji i krzyżowań

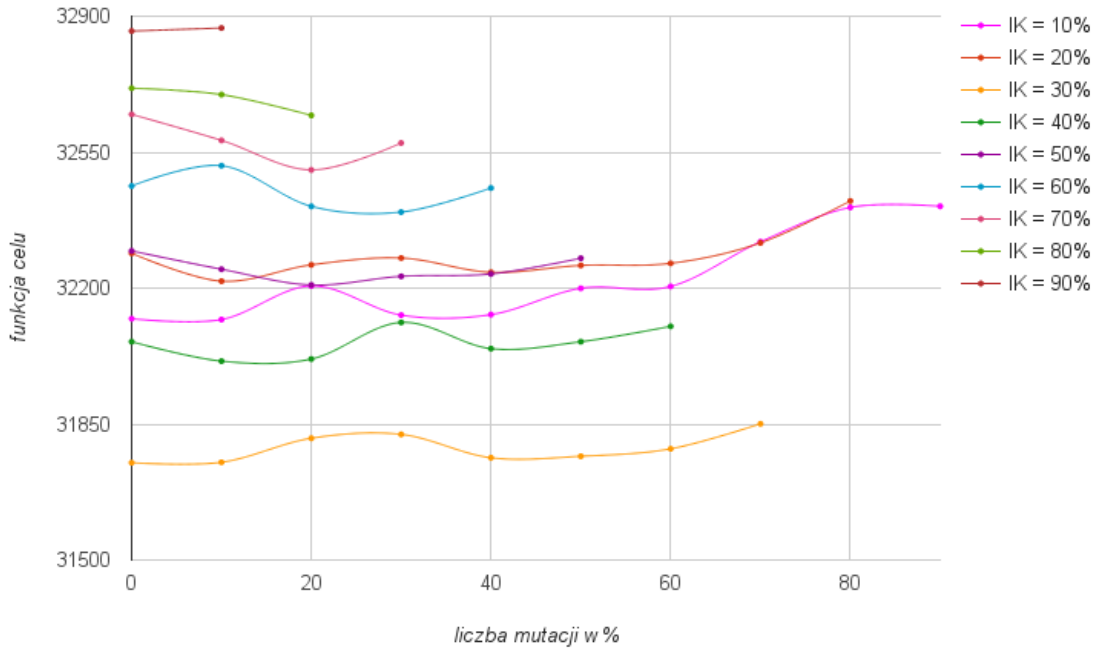
Test ten miał sprawdzić jaka liczba mutacji i krzyżowania wykazuje najlepsze optymalizowanie funkcji celu. Należy tu zauważyć, że $liczba\,mutacji + liczba\,krzyżowań + liczba\,łączonych\,(mutacja + krzyżowanie) = 1$, z tego powodu zawsze będą wykorzystywane obie metody. Na przedstawionym wykresie widać zależności między testowanymi parametrami. Warto zauważyć, że zastosowanie samej mutacji i krzyżowania przynosi gorsze efekty niż z zastosowaniem ich połączenia (punkty pomiarowe dla $IM = 100 - IK[\%]$ przyjmują gorsze wartości niż przy zastosowaniu połączenia obu metod). Najoptymalniejszym rozwiązaniem okazało się zastosowanie krzyżowania w liczbie 30% oraz mutacji w liczbie 10%. Pozostałe 60% stanowi połączenie tych 2 metod.

parametr	SM	PS	PE	IM	IK	T
wartość	$0,15$	250	750	$0-[90-IK]$	$0-0,9$	$8s$

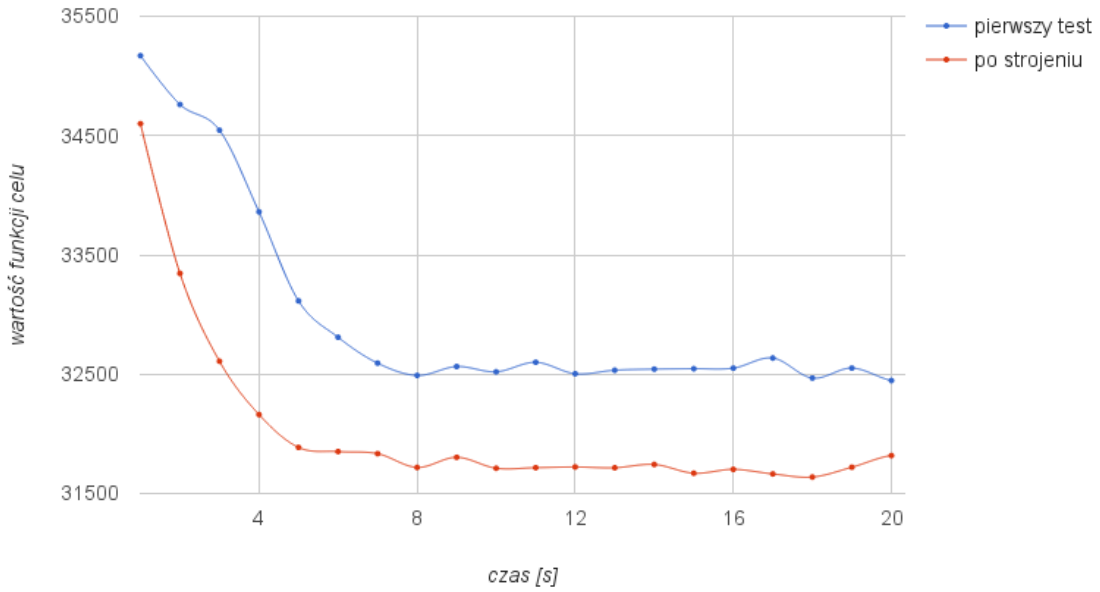
Tablica 7: tabela użytych parametrów dla testu 2.4



Rysunek 4: wykres dla testu wielkości populacji początkowej i końcowej



Rysunek 5: wykres zależności funkcji celu od liczby mutacji i krzyżowania



Rysunek 6: Wykres dla testu wartości funkcji celu od czasu po strojeniu

2.5 Test wartości funkcji celu od czasu po strojeniu

Ten test miał wykazać jak bardzo zmiana parametrów wpłynęła na uzyskiwane wyniki z początkowego testu. Na wykresie widać, że poprawa wartości funkcji celu wyniosła aż 3%. Z wykresu możemy także wyczytać, że algorytm osiągnął optimum po $5s$ (przed strojeniem czas wynosił $8s$).

parametr	SM	PS	PE	IM	IK	T
wartość	$0,15$	250	750	$0,1$	$0,3$	$0-20s$

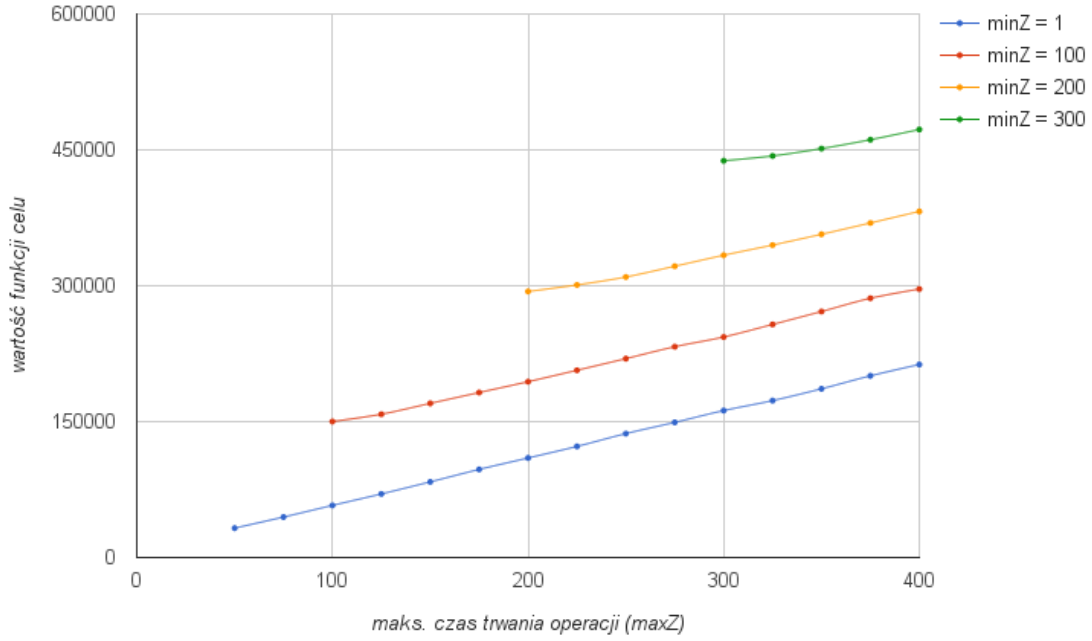
Tablica 8: tabela użytych parametrów dla testu 2.5

3 Testy instancji problemu

Rozdział ten jest przeznaczony na testy dotyczące różnych typów instancji. Ustawienia algorytmu, które zostaną użyte do przeprowadzenia testów pochodzą z poprzedniego rozdziału (tabela 8) i są one najbardziej optymalne dla naszego algorytmu. W dalszej części będziemy używać skrótów opisujących parametry instancji przedstawionych w tabeli 1.

3.1 Czas trwania operacji

Celem testu jest wyznaczenie zależności funkcji celu od długości trwania operacji. Długość ta opisywana jest przez dwie wartości: $minZ$ oraz $maxZ$. Wszystkie parametry instancji zostały przedstawione w tabeli 3.1.
Badaną zależność między funkcją celu a czasem trwania operacji zamieszczono na wykresie 7. Możemy zauważyć, że występuje tutaj zależność liniowa. Prowadząc prostą przez punkty, dla



Rysunek 7: wykres zależności funkcji celu od czasu trwania operacji.

których $minZ = maxZ$ możemy wyznaczyć funkcję opisującą maksymalną wartość funkcji celu w danym przedziale wartości. Na wykresie 8 przedstawiono funkcje liniowe opisujące minimalną i maksymalną wartość funkcji celu w zależności od parametru $maxZ$. Łatwo także zauważyć, że wartości funkcji celu w przedziale $minZ =< 1\,;\, maxZ >$ zmienia się liniowo. Na podstawie tych obserwacji jesteśmy w stanie wyznaczyć wzór opisujący badaną zależność:

$$F_1 = [(y_2(maxZ) - y_1(maxZ)) * \frac{minZ}{maxZ}] + y_1(maxZ) \tag{4}$$

$$y_2 = 1440 * X + 6042 \qquad y_1 = 523 * X + 5202 \tag{5}$$

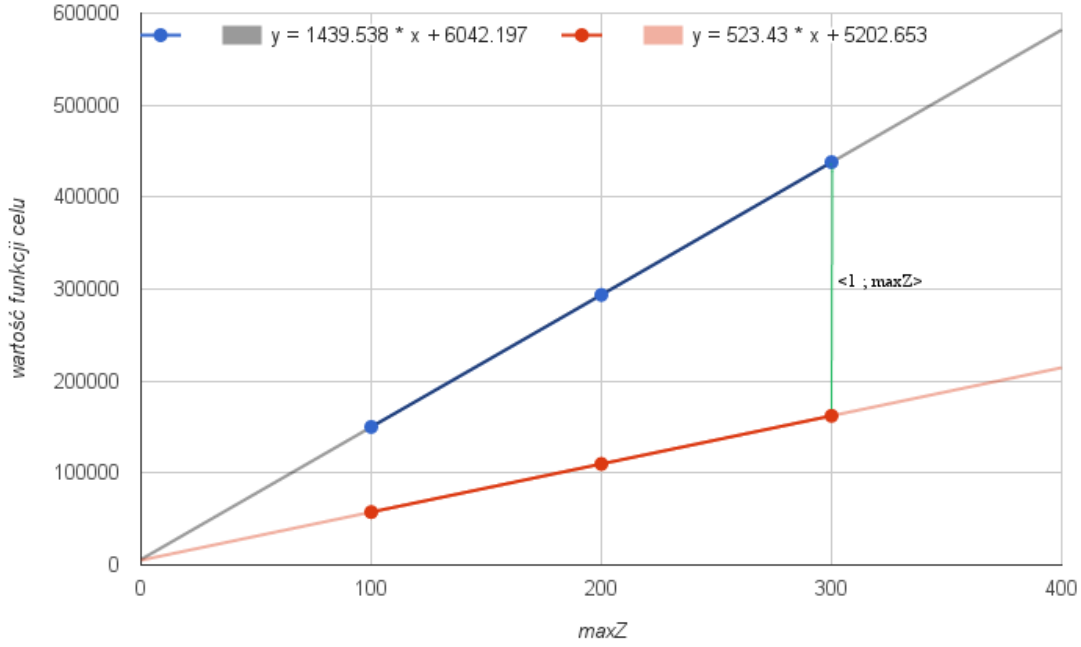
parametr	minZ	maxZ	N	maxG	PP	minP	maxP
wartość	$1 - 400$	$1 - 400$	50	100	$0,2$	1	50

Tablica 9: wartości parametrów dla testu 3.1

3.2 Liczba zadań

Celem tego testu jest sprawdzenie jak liczba zadań wpływa na funkcję celu. Parametry instancji przedstawiono w tabeli 10. Zależność przedstawiono na wykresie 9. Łatwo zauważyć, że wartość funkcji celu jest proporcjonalna do kwadratu liczby zadań. Możemy opisać ją następującym wzorem:

$$F_2 = 14,5 * N^2 - 129 * N + 2643 \tag{6}$$



Rysunek 8: funkcje opisujące minimalną i maksymalną wartość funkcji celu w zależności od $maxZ$.

parametr	minZ	maxZ	N	maxG	PP	minP	maxP
wartość	1	5	$20 - 160$	100	$0,2$	1	50

Tablica 10: wartość parametrów dla testu 3.2

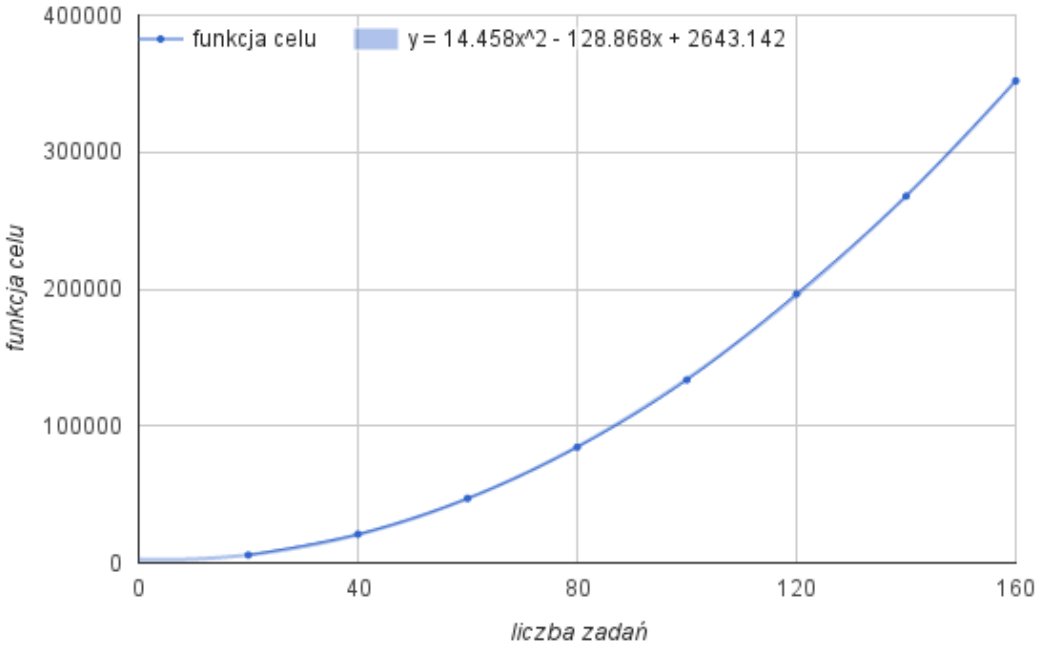
3.3 Czas gotowości dla zadania

Celem zadania jest sprawdzenie jak czas gotowości pojedynczego zadania wpływa na wartość funkcji celu całego rozwiązania. Parametry instancji przedstawiono w tabeli 11, natomiast zależność na wykresie 3.3. Z wykresu możemy odczytać, że wartość funkcji celu jest proporcjonalna do maksymalnego czasu gotowości (zależność liniowa). Prowadząc prostą przez punkty pomiarowe dochodzimy do następującego równania zależności:

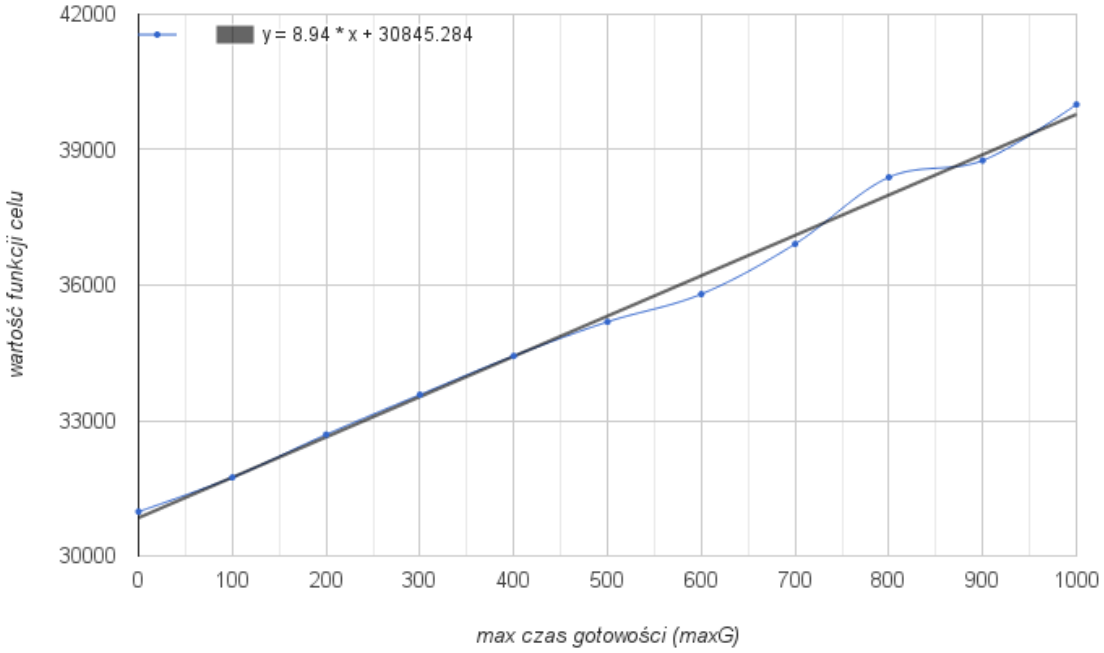
$$F_3 = 9 * X + 30845 \tag{7}$$

parametr	minZ	maxZ	N	maxG	PP	minP	maxP
wartość	1	5	50	0-1000	20%	1	50

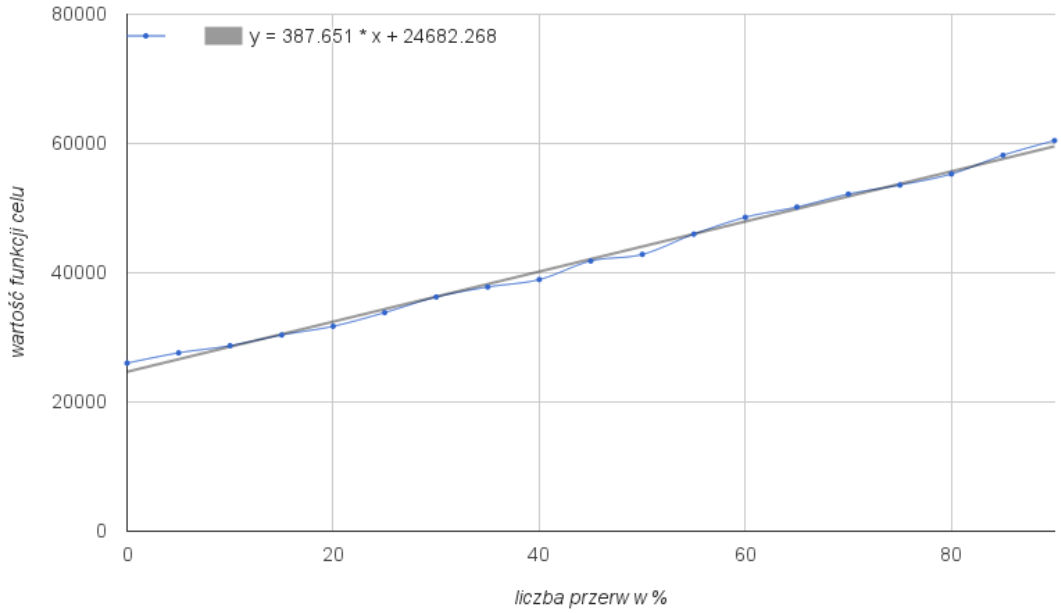
Tablica 11: wartość parametrów dla testu 3.3



Rysunek 9: zależność funkcji celu od ilości zadań N



Rysunek 10: zależność funkcji celu od czasu gotowości zadania



Rysunek 11: zależność funkcji celu od liczby przerw konserwujących

3.4 Liczba przerw konserwujących

Celem rozdziału jest zbadanie jak liczba przerw konserwujących wpływa na wartość funkcji celu. Wartości parametrów zostały przedstawione w tabeli 12, natomiast zależność na wykresie 11. Podobnie jak w rozdziale 3.3 występuje tutaj zależność liniowa, którą możemy opisać wzorem:

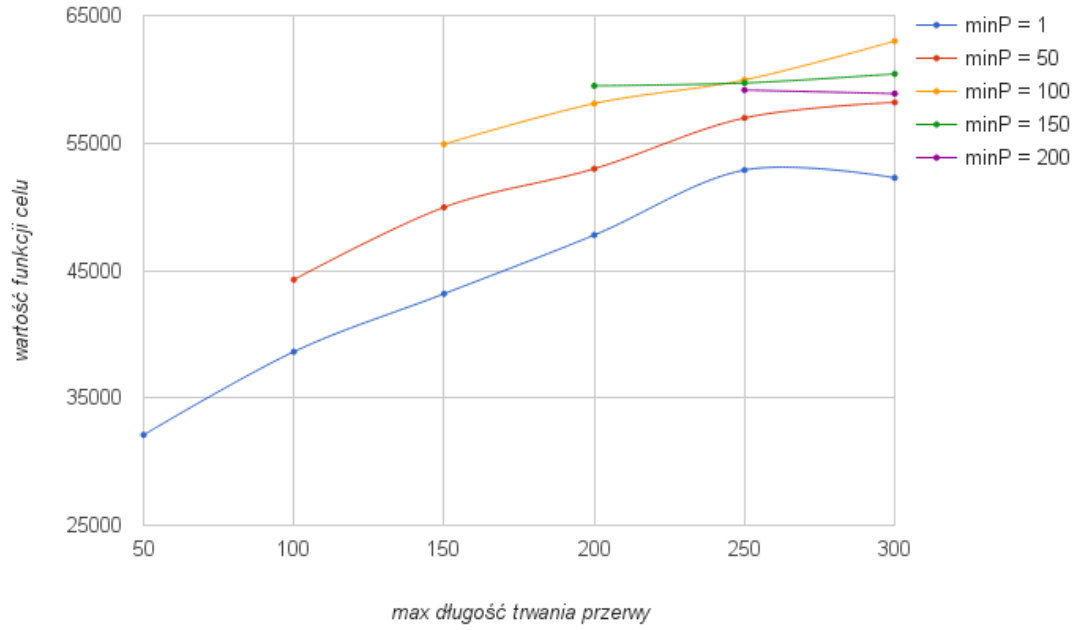
$$F_4 = 388 * PP + 24682 \tag{8}$$

parametr	minZ	maxZ	N	maxG	PP	minP	maxP
wartość	1	5	50	100	$0 - 0,9$	1	50

Tablica 12: wartości parametrów dla testu 3.4

3.5 Czas trwania przerw konserwujących

Celem rozdziału jest zbadanie jak długość przerw konserwujących wpływa na wartość funkcji celu. Wartości parametrów zostały przedstawione w tabeli 13, natomiast zależność na wykresie 12. Możemy tutaj zauważyć pewną ciekawą zależność: duży czas trwania przerw ($minP = \{150, 200\}$), powoduje, że ich wzrost nie wpływa na zmianę wartości funkcji celu. Wytłumaczenie jest proste. Przerwy konserwujące są ustawiane w ten sposób, aby miały znaczenie dla rozwiązanie oraz nie nachodziły na siebie, jednak gdy są bardzo długie, takie założenie staje się niemożliwe (rosną odległości między przerwami, w które mieszczą się zadania). W takim wypadku zadania, których długość jest dużo krótsza od długości przerw uda się uszeregować nim napotkamy na ostatnie przerwy (końcowe przerwy przestają mieć znaczenie).



Rysunek 12: zależność funkcji celu od długości przerw konserwujących

parametr	minZ	maxZ	N	maxG	PP	minP	maxP
wartość	1	5	50	100	0,2	1 – 200	50 – 300

Tablica 13: wartości parametrów dla testu 3.5

4 Podsumowanie

Po przeprowadzeniu wszystkich badań doszliśmy do wniosku, że nasza implementacja uszeregowania spełnia założenia zamieszczone w zadaniu nr 3. Udało nam się zoptymalizować parametry na podstawie wyspecjalizowanych testów, co poprawiło wartość celu o 3%. Optymalizacja skróciła także czas znajdowania optymalnego rozwiązania: przed strojeniem $8s$, po strojeniu $5s$. Poprawa wyników jest zauważalna. Zamieszczono ją na wykresie 2.5, w którym porównano testy przed i po strojeniu. Na podstawie testów instancji (rozdział 3) udało nam się znaleźć zależności między funkcją celu, a poszczególnymi parametrami opisującymi instancję.